Deep Neural Network

LAB 1: - Introduction to various libraries required to implement DNN

Name: - Gaurav Sonawane

PRN: - 20200802154

BTech CSE TY

DS1

In [1]:
```
# Numpy
import numpy as np
from numpy import random
b=np.random.randint(10,100,10)
c=np.random.randint(10,100,10)
print("b -",b)
print("c -",c)
```

```
b - [65 67 27 55 94 77 26 93 54 61]
c - [82 17 12 28 32 46 35 30 61 13]
```

In [2]:
```
# Scipy
from scipy.fftpack import fft, ifft
x= np.array([0,1,2,3])
y= fft(x)
print(y)
```

```
[ 6.-0.j -2.+2.j -2.-0.j -2.-2.j]
```

In [3]:
```
# Sklearn
from sklearn import tree
from sklearn.model_selection import train_test_split
X=[[165,19],[175,32],[136,35],[174,65],[141,28],[176,15],[131,32],[166,6],[128,32],[179,
Y=['Man','Woman','Woman','Man','Woman','Man','Woman','Man','Woman','Man','Woman','Man','
data_feature_names = ['height','length of hair']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state
DTclf = tree.DecisionTreeClassifier()
DTclf = DTclf.fit(X,Y)
prediction = DTclf.predict([[135,29]])
print(prediction)
```

```
['Woman']
```

In [4]:
```
# Tensorflow

import tensorflow as tf
p = tf.constant(10)
q= tf.constant(32)
print(p+q)
```

```
tf.Tensor(42, shape=(), dtype=int32)
```

Loading [MathJax]/extensions/Safe.js

In [5]:

```python
# Pytorch

import torch
t1=torch.tensor([1, 2, 3, 4])
t2=torch.tensor([[1, 2, 3, 4],
                 [5, 6, 7, 8],
                 [9, 10, 11, 12]])
print("Tensor t1: \n", t1)
print("\nTensor t2: \n", t2)
print("\nRank of t1: ", len(t1.shape))
print("Rank of t2: ", len(t2.shape))
print("\nRank of t1: ", t1.shape)
print("Rank of t2: ", t2.shape)
```

```
Tensor t1:
 tensor([1, 2, 3, 4])

Tensor t2:
 tensor([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]])

Rank of t1:  1
Rank of t2:  2

Rank of t1:  torch.Size([4])
Rank of t2:  torch.Size([3, 4])
```

In [6]:

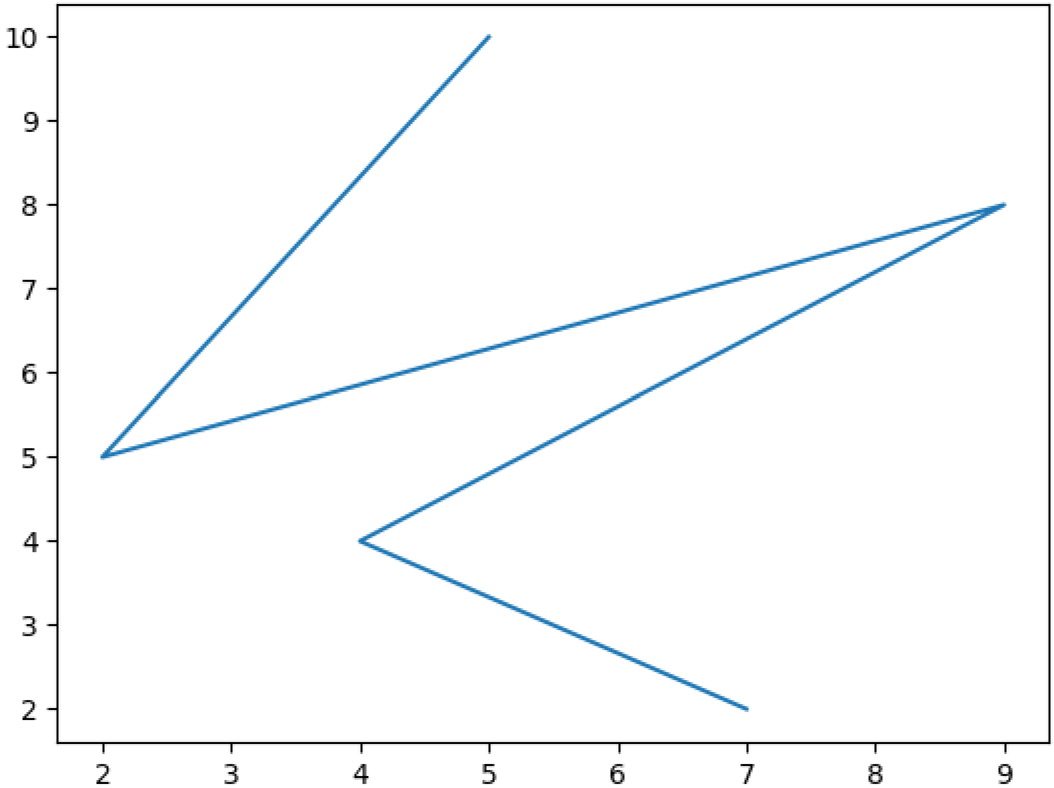
```python
# Pandas

import pandas as pd
ser = pd.Series()
print(ser)
data = np.array(['h','e','l','l','o'])
ser = pd.Series(data)
print(ser)
```

```
Series([], dtype: float64)
0    h
1    e
2    l
3    l
4    o
dtype: object
```

```
<ipython-input-6-424a8a1c87cb>:4: FutureWarning: The default dtype for empty Series will
be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to sile
nce this warning.
  ser = pd.Series()
```

In [7]:

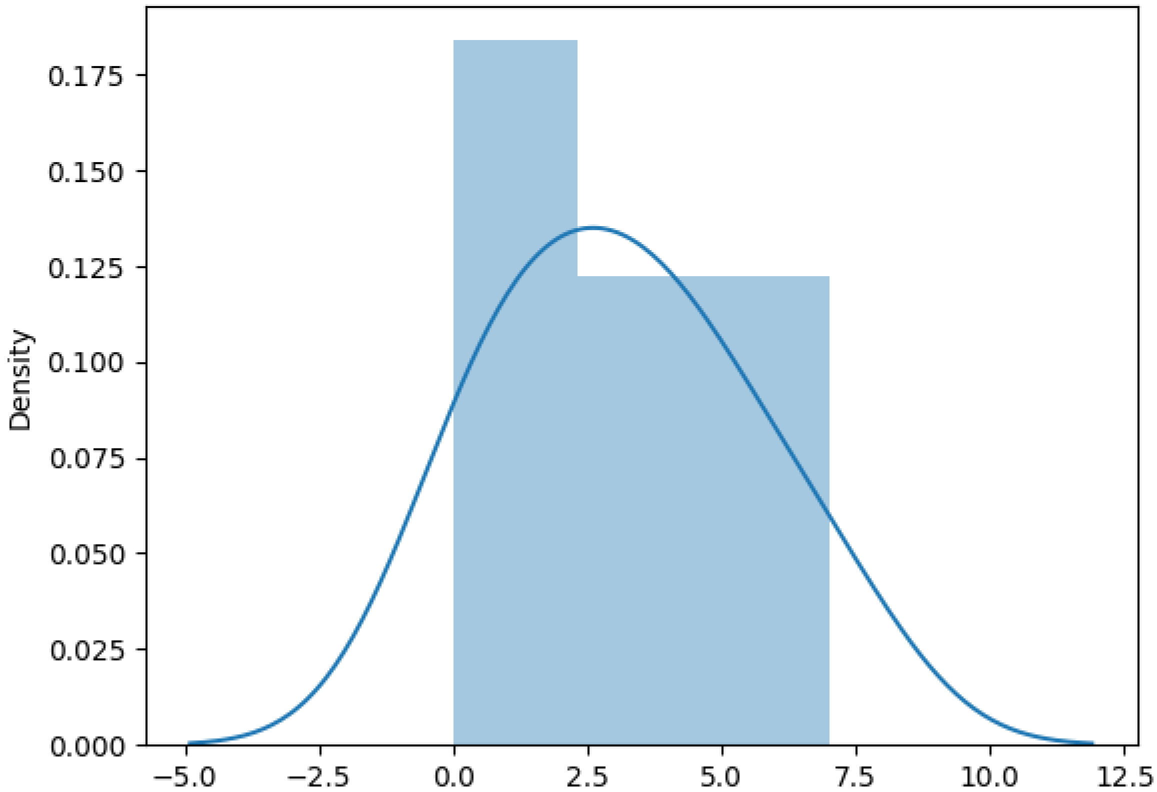
```python
# Matplotlib

from matplotlib import pyplot as plt
x = [5, 2, 9, 4, 7]
y = [10, 5, 8, 4, 2]
plt.plot(x,y)
plt.show()
```



In [8]:

```
# Seaborn

import seaborn as sns
sns.distplot([0, 1, 4, 3, 2, 5, 7])
plt.show()
```

```
<ipython-input-8-c5ae3dcb91c9>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot([0, 1, 4, 3, 2, 5, 7])
```


Density
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5

Deep Neural Network

LAB 2: PCA

NAME : - Gaurav Sonawane

PRN : - 20200802154

In [4]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [5]:
```
df=pd.read_csv("S:/3rd Year/SEM VI/TC3 DNN/LAB/LAB 2/mnist_data.csv")
df.head()
```

Out[5]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pix
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

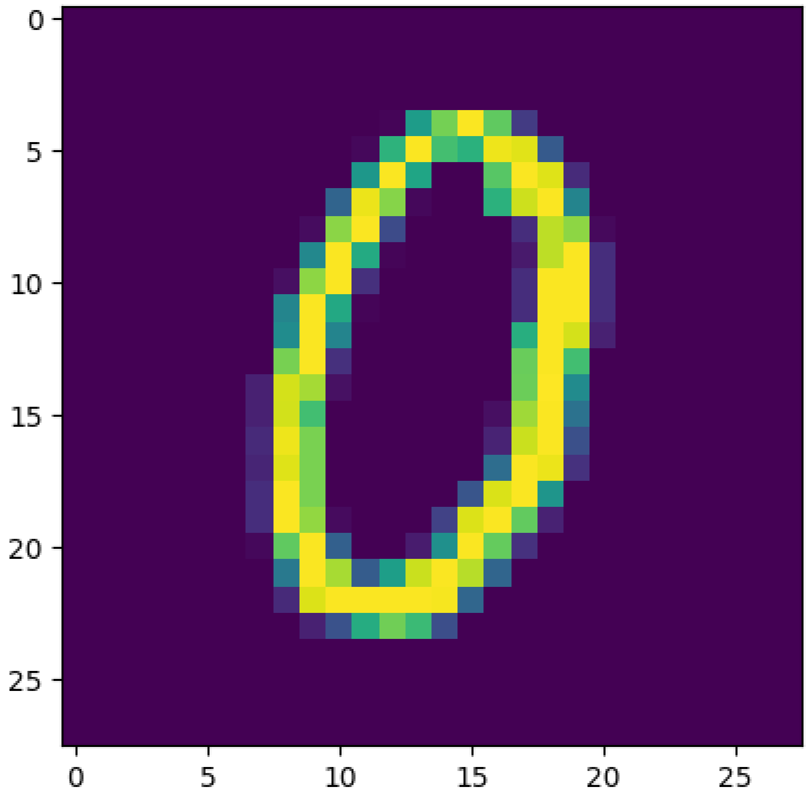
5 rows × 785 columns

In [6]:
```
df.shape
```

Out[6]:
```
(42000, 785)
```

In [7]:
```
plt.imshow(df.iloc[5,1:].values.reshape(28,28))
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x28c7ff19990>
```



In [8]:
```
x=df.iloc[:,1:]
y=df.iloc[:,0]
```

In [9]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)
```

In [10]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
x_train=ss.fit_transform(x_train)
x_test=ss.transform(x_test)
```

In [11]:
```
cov_matrix = np.cov(x_train.T)

# Compute the eigenvectors and eigenvalues
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
cov_matrix
```

Out[11]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [12]:
```
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
eigenvalues
```

```
Out[12]: array([ 4.06711120e+01,  2.91702340e+01,  2.67445962e+01,  2.08534479e+01,
        1.81489188e+01,  1.58529825e+01,  1.38710810e+01,  1.24805897e+01,
        1.10279424e+01,  1.00958253e+01,  9.63317821e+00,  8.62785945e+00,
        8.06303131e+00,  7.89511749e+00,  7.44167929e+00,  7.17032873e+00,
        6.73266373e+00,  6.62744023e+00,  6.41499161e+00,  6.25808269e+00,
        5.90495742e+00,  5.76521585e+00,  5.52084601e+00,  5.32003847e+00,
        5.18309925e+00,  4.93439597e+00,  4.90652171e+00,  4.71800493e+00,
        4.49824444e+00,  4.43140305e+00,  4.32604521e+00,  4.23491831e+00,
        4.10335015e+00,  4.06731180e+00,  4.02362178e+00,  3.84130473e+00,
        3.81886146e+00,  3.71316498e+00,  3.60918108e+00,  3.47303214e+00,
        3.42842989e+00,  3.38841159e+00,  3.29157511e+00,  3.22927379e+00,
        3.21745142e+00,  3.15810372e+00,  3.12620905e+00,  3.10412385e+00,
        3.05892795e+00,  3.03728188e+00,  2.96540461e+00,  2.94033669e+00,
        2.86828564e+00,  2.82617865e+00,  2.80038441e+00,  2.77411221e+00,
        2.71978838e+00,  2.69394232e+00,  2.64724369e+00,  2.63065837e+00,
        2.56498699e+00,  2.53651735e+00,  2.48789582e+00,  2.44610051e+00,
        2.42066567e+00,  2.37577156e+00,  2.35505085e+00,  2.33408709e+00,
        2.29804042e+00,  2.25323781e+00,  2.24106526e+00,  2.18913049e+00,
        2.17678604e+00,  2.15005558e+00,  2.13513807e+00,  2.11873568e+00,
        2.09860149e+00,  2.08173945e+00,  2.04964573e+00,  2.03090509e+00,
        2.02167211e+00,  2.01348141e+00,  1.98804025e+00,  1.97986315e+00,
        1.97403678e+00,  1.94691442e+00,  1.92889204e+00,  1.90355211e+00,
        1.89071057e+00,  1.86950768e+00,  1.85666825e+00,  1.84729644e+00,
        1.82895483e+00,  1.80723939e+00,  1.79915564e+00,  1.79257345e+00,
        1.76700793e+00,  1.76291349e+00,  1.73547115e+00,  1.71657552e+00,
        1.68838208e+00,  1.67856907e+00,  1.65450839e+00,  1.63633366e+00,
        1.62408775e+00,  1.61753416e+00,  1.58633862e+00,  1.57931566e+00,
        1.57457617e+00,  1.56247777e+00,  1.53703382e+00,  1.52553601e+00,
        1.50578195e+00,  1.49456642e+00,  1.46197643e+00,  1.44645496e+00,
        1.44165522e+00,  1.41256843e+00,  1.40546030e+00,  1.39995146e+00,
        1.37660517e+00,  1.37344507e+00,  1.35436200e+00,  1.34695707e+00,
        1.33286787e+00,  1.31897410e+00,  1.31553999e+00,  1.29903719e+00,
        1.29464689e+00,  1.28446097e+00,  1.26255597e+00,  1.25092334e+00,
        1.23641911e+00,  1.21799879e+00,  1.21357858e+00,  1.20903264e+00,
        1.18946022e+00,  1.18428989e+00,  1.16634459e+00,  1.15923589e+00,
        1.14433496e+00,  1.12663352e+00,  1.12405960e+00,  1.11700290e+00,
        1.11127079e+00,  1.10017688e+00,  1.08105822e+00,  1.07682637e+00,
        1.06479267e+00,  1.05699736e+00,  1.04481442e+00,  1.03615148e+00,
        1.03180265e+00,  1.02632173e+00,  1.02397615e+00,  1.01989409e+00,
        1.00383881e+00,  1.00127069e+00,  9.97426506e-01,  9.96929229e-01,
        9.94484639e-01,  9.89654036e-01,  9.79400305e-01,  9.77081867e-01,
        9.73776975e-01,  9.58117658e-01,  9.53727214e-01,  9.43577486e-01,
        9.42442675e-01,  9.35207696e-01,  9.30559812e-01,  9.15491484e-01,
        9.07369233e-01,  9.05380397e-01,  8.97880716e-01,  8.86720060e-01,
        8.76218812e-01,  8.69039317e-01,  8.61501794e-01,  8.51985338e-01,
        8.46801205e-01,  8.32716901e-01,  8.23530433e-01,  8.14690826e-01,
        8.06889664e-01,  7.93623433e-01,  7.92750208e-01,  7.83390002e-01,
        7.74205208e-01,  7.64506660e-01,  7.60577097e-01,  7.58300582e-01,
        7.51686091e-01,  7.44858803e-01,  7.43025993e-01,  7.37756397e-01,
        7.28100581e-01,  7.20804662e-01,  7.11609067e-01,  7.10259445e-01,
        7.00498504e-01,  6.93296508e-01,  6.88506838e-01,  6.83911784e-01,
        6.83695771e-01,  6.80593710e-01,  6.73411643e-01,  6.64650059e-01,
        6.56889415e-01,  6.42502188e-01,  6.38809226e-01,  6.36149508e-01,
        6.27898833e-01,  6.25320344e-01,  6.18852112e-01,  6.10213226e-01,
        5.97298411e-01,  5.95319971e-01,  5.87937633e-01,  5.78113643e-01,
        5.75525903e-01,  5.72695215e-01,  5.69323427e-01,  5.64862404e-01,
        5.61615225e-01,  5.49234289e-01,  5.47933629e-01,  5.42187373e-01,
        5.33852001e-01,  5.28430966e-01,  5.25367410e-01,  5.21060016e-01,
        5.17532115e-01,  5.12563365e-01,  5.08250861e-01,  5.02010390e-01,
        4.96889997e-01,  4.93255184e-01,  4.89672681e-01,  4.87552462e-01,
        4.82898213e-01,  4.74861075e-01,  4.71885677e-01,  4.65202079e-01,
        4.63237312e-01,  4.57944268e-01,  4.49498936e-01,  4.43655098e-01,
        4.40422461e-01,  4.40072972e-01,  4.34513409e-01,  4.32123305e-01,
        4.28242912e-01,  4.23123714e-01,  4.18504719e-01,  4.12356493e-01,
```

```
4.10445934e-01,  4.09313851e-01,  4.04486101e-01,  4.01602179e-01,
3.95419372e-01,  3.94304905e-01,  3.89926276e-01,  3.86656589e-01,
3.84181196e-01,  3.80779135e-01,  3.77162383e-01,  3.75563600e-01,
3.73961900e-01,  3.73145479e-01,  3.66827471e-01,  3.63212280e-01,
3.62715972e-01,  3.58142585e-01,  3.54548625e-01,  3.53667339e-01,
3.51140377e-01,  3.50590764e-01,  3.46467537e-01,  3.39558446e-01,
3.38169318e-01,  3.35575551e-01,  3.33503840e-01,  3.30622615e-01,
3.24971946e-01,  3.23770795e-01,  3.21392322e-01,  3.19059274e-01,
3.16686967e-01,  3.11622921e-01,  3.10324882e-01,  3.06461004e-01,
3.02377049e-01,  2.98224193e-01,  2.97234560e-01,  2.93906200e-01,
2.91201161e-01,  2.89101835e-01,  2.88305179e-01,  2.84875369e-01,
2.83461842e-01,  2.80553304e-01,  2.79588059e-01,  2.77723182e-01,
2.77076153e-01,  2.71408458e-01,  2.69756166e-01,  2.67785646e-01,
2.66128714e-01,  2.64844935e-01,  2.60580373e-01,  2.60214060e-01,
2.55565773e-01,  2.51718667e-01,  2.50106125e-01,  2.49170336e-01,
2.48614796e-01,  2.46173131e-01,  2.42020188e-01,  2.39990920e-01,
2.38933996e-01,  2.36476259e-01,  2.35145624e-01,  2.34124830e-01,
2.32821354e-01,  2.32255625e-01,  2.31320637e-01,  2.29134185e-01,
2.25277070e-01,  2.24769863e-01,  2.23439838e-01,  2.20961370e-01,
2.19363959e-01,  2.18426955e-01,  2.17921443e-01,  2.16537090e-01,
2.13632795e-01,  2.12091662e-01,  2.10369928e-01,  2.09461504e-01,
2.08871733e-01,  2.06658499e-01,  2.03110451e-01,  2.01690866e-01,
2.00228898e-01,  1.98806485e-01,  1.98112318e-01,  1.96148946e-01,
1.94521338e-01,  1.92581263e-01,  1.92111648e-01,  1.91156875e-01,
1.89144127e-01,  1.88372021e-01,  1.85328372e-01,  1.84557700e-01,
1.82613374e-01,  1.80720387e-01,  1.79451660e-01,  1.77853807e-01,
1.77326433e-01,  1.76099598e-01,  1.75323989e-01,  1.74524905e-01,
1.73646747e-01,  1.72318837e-01,  1.71109230e-01,  1.69903874e-01,
1.69320679e-01,  1.67280554e-01,  1.67215670e-01,  1.64527044e-01,
1.64401886e-01,  1.63378012e-01,  1.61550085e-01,  1.59319656e-01,
1.58385751e-01,  1.57744327e-01,  1.57437644e-01,  1.55462113e-01,
1.55298696e-01,  1.54393992e-01,  1.53396259e-01,  1.50859289e-01,
1.50162255e-01,  1.48614075e-01,  1.47365174e-01,  1.46833168e-01,
1.46587465e-01,  1.44740263e-01,  1.44035610e-01,  1.43507830e-01,
1.42527662e-01,  1.41048195e-01,  1.39938032e-01,  1.37970959e-01,
1.37560122e-01,  1.37018192e-01,  1.35891165e-01,  1.35536843e-01,
1.34432524e-01,  1.33105325e-01,  1.32978207e-01,  1.31490331e-01,
1.30607749e-01,  1.29205857e-01,  1.28972387e-01,  1.27805979e-01,
1.27592465e-01,  1.27321761e-01,  1.26397051e-01,  1.24786798e-01,
1.24269635e-01,  1.23694041e-01,  1.23217484e-01,  1.22322447e-01,
1.22113652e-01,  1.20910437e-01,  1.20617410e-01,  1.20175752e-01,
1.19958395e-01,  1.18577036e-01,  1.17614173e-01,  1.16940761e-01,
1.16223963e-01,  1.15798521e-01,  1.15264262e-01,  1.14513234e-01,
1.13534434e-01,  1.13260182e-01,  1.12787042e-01,  1.12381743e-01,
1.11124150e-01,  1.10145829e-01,  1.10050040e-01,  1.09616324e-01,
1.08348935e-01,  1.08300706e-01,  1.06973631e-01,  1.06316585e-01,
1.05616439e-01,  1.05011397e-01,  1.04572941e-01,  1.03564765e-01,
1.03412050e-01,  1.02839928e-01,  1.01636819e-01,  1.01255458e-01,
1.01105906e-01,  1.00514976e-01,  1.00184284e-01,  9.96684830e-02,
9.85944138e-02,  9.81792268e-02,  9.78504378e-02,  9.72584135e-02,
9.68934512e-02,  9.62566772e-02,  9.59103477e-02,  9.52237233e-02,
9.49389365e-02,  9.46583732e-02,  9.40632282e-02,  9.35594675e-02,
9.27343210e-02,  9.23096116e-02,  9.19672269e-02,  9.09599392e-02,
9.04369172e-02,  9.01924150e-02,  8.98204211e-02,  8.91661923e-02,
8.87454590e-02,  8.79511694e-02,  8.75219487e-02,  8.71315811e-02,
8.67481398e-02,  8.64379862e-02,  8.63835135e-02,  8.52396640e-02,
8.50460084e-02,  8.46017000e-02,  8.45216562e-02,  8.37672270e-02,
8.34251801e-02,  8.32407267e-02,  8.26291606e-02,  8.16523405e-02,
8.12074436e-02,  8.07819985e-02,  8.05980306e-02,  7.99180174e-02,
7.94910881e-02,  7.87843828e-02,  7.83714765e-02,  7.80096672e-02,
7.77109551e-02,  7.72753449e-02,  7.70754197e-02,  7.66766659e-02,
7.61279972e-02,  7.57607480e-02,  7.56469532e-02,  7.49928889e-02,
7.48413790e-02,  7.44274341e-02,  7.40097818e-02,  7.38105967e-02,
7.34247948e-02,  7.28316794e-02,  7.27521076e-02,  7.20681142e-02,
```

```
7.17179983e-02,  7.10920617e-02,  7.08459734e-02,  7.06143542e-02,
7.01964058e-02,  6.94111995e-02,  6.93626571e-02,  6.87923709e-02,
6.86738722e-02,  6.84332040e-02,  6.81816058e-02,  6.77511150e-02,
6.71123773e-02,  6.68630720e-02,  6.66564923e-02,  6.63671805e-02,
6.60491787e-02,  6.58188216e-02,  6.54857986e-02,  6.50664817e-02,
6.43288568e-02,  6.39321809e-02,  6.38124529e-02,  6.32487261e-02,
6.30981736e-02,  6.27935069e-02,  6.22230899e-02,  6.19232021e-02,
6.12613083e-02,  6.09097469e-02,  6.05987073e-02,  6.05811147e-02,
6.02415501e-02,  5.98497587e-02,  5.93304647e-02,  5.90873281e-02,
5.88759811e-02,  5.84372660e-02,  5.83566800e-02,  5.79181109e-02,
5.74497859e-02,  5.69782897e-02,  5.67421110e-02,  5.63601195e-02,
5.61221112e-02,  5.56105972e-02,  5.52836678e-02,  5.52679809e-02,
5.48752293e-02,  5.47999442e-02,  5.39675343e-02,  5.37684365e-02,
5.36751826e-02,  5.33642190e-02,  5.30602825e-02,  5.27956592e-02,
5.24920724e-02,  5.22479871e-02,  5.18142369e-02,  5.14492449e-02,
5.12397050e-02,  5.11116882e-02,  5.07736540e-02,  5.06580447e-02,
4.99980383e-02,  4.95403951e-02,  4.93733741e-02,  4.91105882e-02,
4.89780727e-02,  4.86829738e-02,  4.84215312e-02,  4.82002997e-02,
4.79783270e-02,  4.76974442e-02,  4.74796657e-02,  4.71005459e-02,
4.68010667e-02,  4.64569782e-02,  4.63805838e-02,  4.59679357e-02,
4.53549621e-02,  4.52183119e-02,  4.50946263e-02,  4.49829410e-02,
4.46594127e-02,  4.41333831e-02,  4.41017729e-02,  4.39301028e-02,
4.35730646e-02,  4.33532780e-02,  4.31228787e-02,  4.28758656e-02,
4.24557960e-02,  4.23441759e-02,  4.22520872e-02,  4.20158859e-02,
4.16316165e-02,  4.15030841e-02,  4.12139915e-02,  4.09837753e-02,
4.04145818e-02,  4.01952845e-02,  4.00283808e-02,  4.00020753e-02,
3.97936747e-02,  3.94828039e-02,  3.93847027e-02,  3.91064287e-02,
3.89694961e-02,  3.88011099e-02,  3.85918057e-02,  3.83147844e-02,
3.81599478e-02,  3.78092570e-02,  3.76500011e-02,  3.74136885e-02,
3.71797145e-02,  3.68429948e-02,  3.66300626e-02,  3.62156971e-02,
3.61121272e-02,  3.58524641e-02,  3.56004879e-02,  3.54469566e-02,
3.52025550e-02,  3.49341396e-02,  3.47748375e-02,  3.44038587e-02,
3.42604028e-02,  3.39124762e-02,  3.38173724e-02,  3.36841400e-02,
3.28397687e-02,  3.26201914e-02,  3.24693271e-02,  3.23908574e-02,
3.21660994e-02,  3.18111020e-02,  3.15645925e-02,  3.13856596e-02,
3.13107287e-02,  3.10984669e-02,  3.06697419e-02,  3.05193018e-02,
3.03289162e-02,  2.99446756e-02,  2.95197910e-02,  2.92750251e-02,
2.91545179e-02,  2.89633973e-02,  2.85548931e-02,  2.82076275e-02,
2.81258221e-02,  2.79152768e-02,  2.74927546e-02,  2.73833960e-02,
2.73173399e-02,  2.68607579e-02,  2.66787713e-02,  2.64066608e-02,
2.61316894e-02,  2.60313366e-02,  2.58209125e-02,  2.53165856e-02,
2.50528688e-02,  2.49047482e-02,  2.47375705e-02,  2.43887731e-02,
2.40203055e-02,  2.38313372e-02,  2.35669430e-02,  2.32960733e-02,
2.29659671e-02,  2.28820808e-02,  2.26013715e-02,  2.19403098e-02,
2.13312466e-02,  2.09341572e-02,  2.06333970e-02,  1.99636390e-02,
1.83367665e-02,  1.78679427e-02,  1.05638969e-02,  7.64010936e-03,
1.74402805e-03,  3.92630346e-15,  3.28102227e-15,  7.02224031e-16,
9.28776748e-17,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
```

```
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00, -9.55494532e-17, -3.13100654e-16])
```

In [13]:
```
from sklearn.decomposition import PCA
pca=PCA(n_components=200)
```

In [14]:
```
x_train_trf=pca.fit_transform(x_train)
x_test_trf=pca.transform(x_test)
```

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(x_train_trf,y_train)

from sklearn.metrics import accuracy_score
y_pred=knn.predict(x_test_trf)
accuracy_score(y_test,y_pred)*100
```

Out[15]:
```
95.03571428571429
```

In [16]:
```
pca=PCA(n_components=3)
x_train_trf=pca.fit_transform(x_train)
x_test_trf=pca.transform(x_test)
x_train_trf
```

Out[16]:
```
array([[-2.71863724, -0.48980201,  1.13561608],
       [-0.67695544, -6.75367743, -2.33555767],
       [-3.03323189,  6.50991056,  7.49191501],
       ...,
       [ 2.14881968,  0.78039773, -0.74814007],
       [ 1.05955935,  0.94781101,  3.94981425],
       [17.70256155,  1.96177427, -4.94391761]])
```

In [18]:
```
import plotly.express as px
y_train_trf=y_train.astype(str)
fig=px.scatter_3d(x=x_train_trf[:,0],
               y=x_train_trf[:,1],
               z=x_train_trf[:,2],
              color=y_train_trf,
               color_discrete_sequence=px.colors.qualitative.G10)
fig.show()
```

In [19]:
```
pca.explained_variance_  # eigen values
```

Out[19]:
```
array([40.67111198, 29.17023395, 26.7445959 ])
```

In [20]:
```
pca.components_.shape # eigen vectors
```

Out[20]:
```
(3, 784)
```

In [21]:
```
pca.explained_variance_ratio_
```

Out[21]:
```
array([0.05785192, 0.0414927 , 0.03804239])
```

Deep Neural Network

LAB 3: - Getting Familiar with tensorflow.

Name: - Gaurav Sonawane

PRN: - 20200802154

BTech CSE TY

DS1

In [1]:
```
# Importing required libraries

import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

In [2]:
```
# Read the dataset
df=pd.read_csv("/content/Churn Dataset.csv")
df=pd.DataFrame(df)
df.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic

5 rows × 21 columns

In [3]:
```
df.Churn.replace(["Yes", "No"], [1, 0], inplace= True)
```

In [4]:
```
df=pd.get_dummies(df)
```

In [5]:
```
x=df.drop(columns=['Churn'])
y=df['Churn']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [6]:
```
# Importing required libraries
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score
```

In [7]:
```
model = Sequential()
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [8]:
```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [9]:
```
model.fit(x_train, y_train, epochs=10, batch_size=10)
```

```
Epoch 1/10
564/564 [==============================] - 3s 4ms/step - loss: 0.4773 - accuracy: 0.7760
Epoch 2/10
564/564 [==============================] - 3s 5ms/step - loss: 0.4220 - accuracy: 0.8030
Epoch 3/10
564/564 [==============================] - 2s 4ms/step - loss: 0.4041 - accuracy: 0.8135
Epoch 4/10
564/564 [==============================] - 2s 4ms/step - loss: 0.3813 - accuracy: 0.8277
Epoch 5/10
564/564 [==============================] - 2s 4ms/step - loss: 0.3522 - accuracy: 0.8470
Epoch 6/10
564/564 [==============================] - 3s 5ms/step - loss: 0.3031 - accuracy: 0.8809
Epoch 7/10
564/564 [==============================] - 3s 5ms/step - loss: 0.2369 - accuracy: 0.9295
Epoch 8/10
564/564 [==============================] - 2s 4ms/step - loss: 0.1585 - accuracy: 0.9728
Epoch 9/10
564/564 [==============================] - 2s 4ms/step - loss: 0.0862 - accuracy: 0.9924
Epoch 10/10
564/564 [==============================] - 2s 4ms/step - loss: 0.0454 - accuracy: 0.9979
```

Out[9]:
```
<keras.callbacks.History at 0x7fcb936b7220>
```

In [10]:
```
y_hat=model.predict(x_test)
y_hat={ 0 if val<0 else 1 for val in y_hat  }
```

```
45/45 [==============================] - 0s 2ms/step
```

In [11]:
```
accuracy_score= model.evaluate(x_test, y_test)
```

```
45/45 [==============================] - 0s 3ms/step - loss: 0.4906 - accuracy: 0.7566
```

Deep Neural Network

LAB 4: -Implementation of Linear Regression with tensorflow.

Name: - Gaurav Sonawane

PRN: - 20200802154

BTech CSE TY

DS1

In [17]:
```
#Import imp Libraries
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

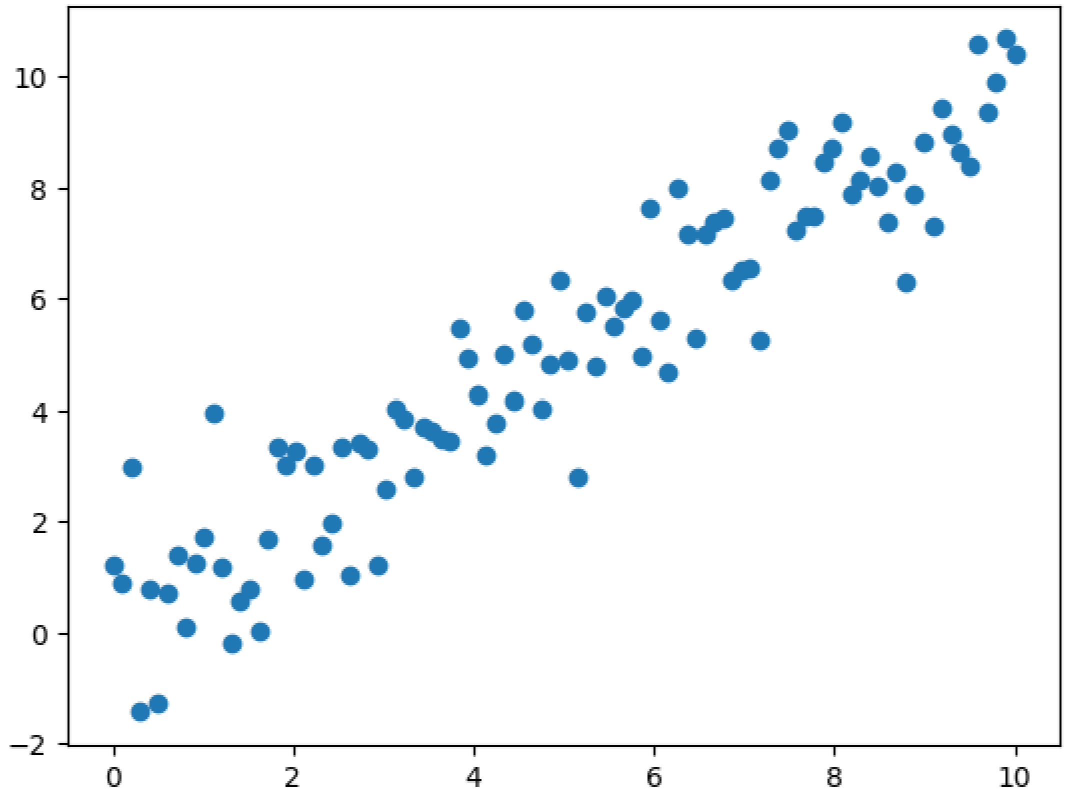
In [18]:
```
# Learning Rate
learning_rate = 0.01
```

In [19]:
```
# Numbers of loop for training through all your data to up[date the parameters
training_epochs = 100
```

In [20]:
```
# The Training Dataset
x_train = np.linspace(0,10,100)
y_train = x_train + np.random.normal(0, 1, 100)
```

In [21]:
```
# Plot of data
plt.scatter(x_train,y_train)
```

Out[21]:
```
<matplotlib.collections.PathCollection at 0x7f952e72b2e0>
```



In [22]:
```
# Declare weight
weight = tf.Variable(0.)
bias = tf.Variable(0.)
```

In [23]:
```
# Define Linear Regression

def line_reg(x):
  y = weight*x + bias
  return y
```

In [24]:
```
# Define loss function (MSE)
def squared_error(y_pred, y_true):
  return tf.reduce_mean(tf.square(y_pred - y_true))
```

In [25]:
```
# train model
for epoch in range(training_epochs):

# Compute loss within Gradient Tape context
  with tf.GradientTape() as tape:
    y_predicted = line_reg(x_train)
    loss = squared_error(y_predicted, y_train)

# Get gradients
gradients = tape.gradient(loss, [weight,bias])

# Adjust weights
weight.assign_sub(gradients[0]*learning_rate)
bias.assign_sub(gradients[1]*learning_rate)

# Print output
print(f"Epoch count {epoch}: Loss value: {loss.numpy()}")
```

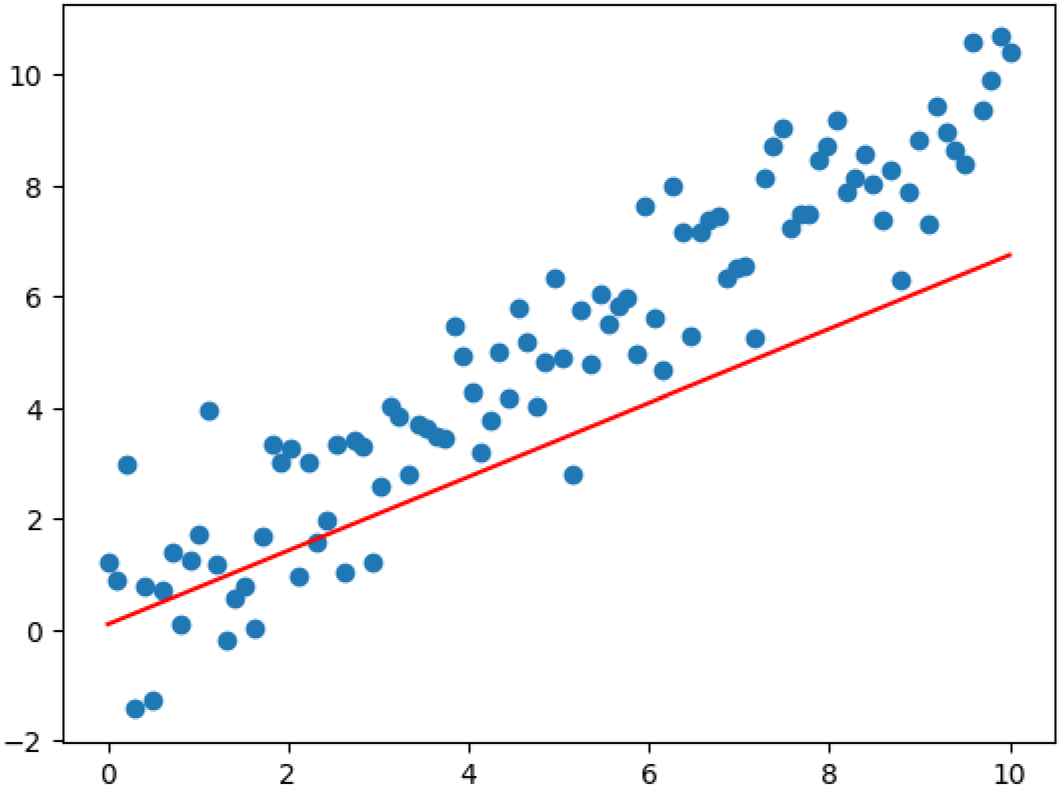
```
Epoch count 99: Loss value: 34.01313400268555
```

In [26]:
```
print(weight.numpy())
print(bias.numpy())
```

Loading [MathJax]/extensions/Safe.js

```
0.6646843
0.100334406
```

In [27]:
```
# Plot the best fit line
plt.scatter(x_train, y_train)
plt.plot(x_train, line_reg(x_train), 'r')
plt.show()
```



Deep Neural Network

LAB 5: -Tensorflow implementation of logistic regression.

Name: - Gaurav Sonawane

PRN: - 20200802154

BTech CSE TY

DS1

In [1]:

```
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
```

```
WARNING:tensorflow:From /usr/local/lib/python3.9/dist-packages/tensorflow/python/compat/
v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_scope)
is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:

```
data = pd.read_csv('/content/data.csv', header = None)
print(data.head())
```

```
   0    1    2  3
0  0  5.1  3.5  1
1  1  4.9  3.0  1
2  2  4.7  3.2  1
3  3  4.6  3.1  1
4  4  5.0  3.6  1
```

In [3]:

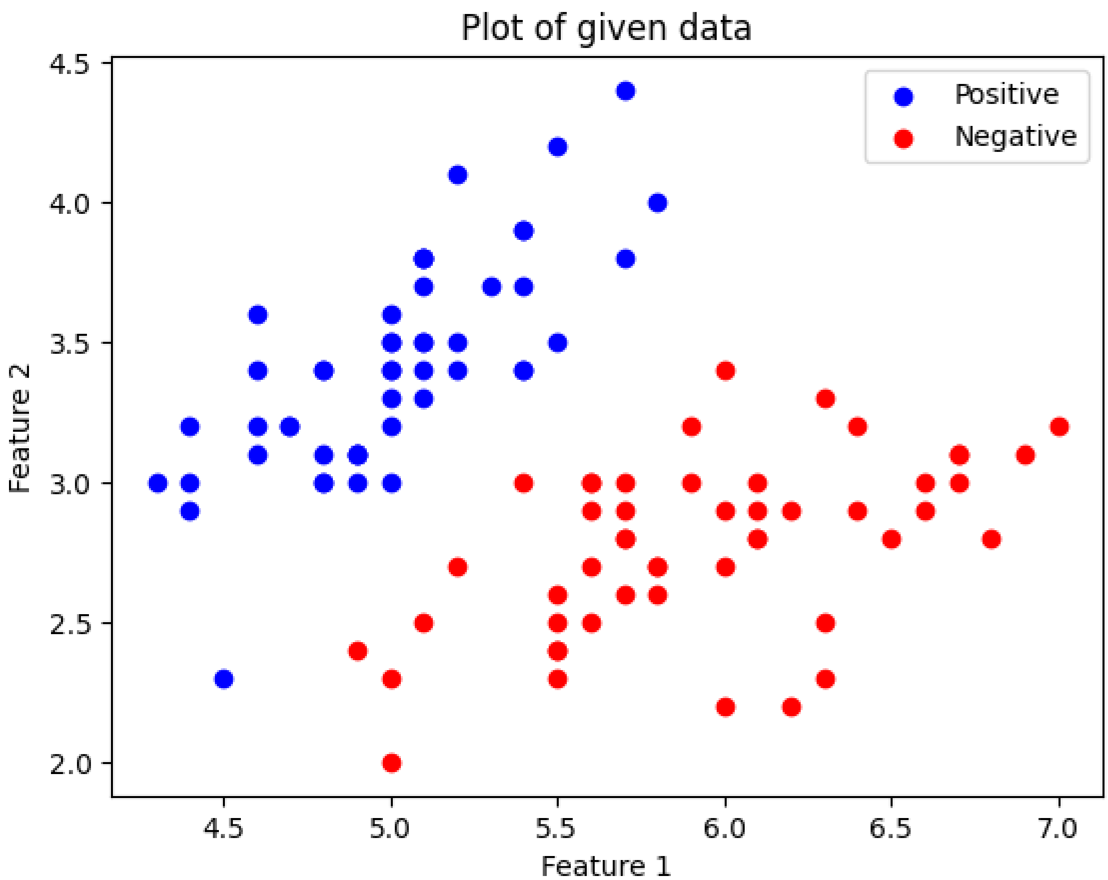
```
x_orig = data.iloc[:,1:-1].values

# Data labels
y_orig = data.iloc[:,-1:].values

print("Shape of Feature Matrix:", x_orig.shape)
print("Shape Label Vector:", y_orig.shape)
```

```
Shape of Feature Matrix: (100, 2)
Shape Label Vector: (100, 1)
```

In [4]:

```
x_pos = np.array([x_orig[i] for i in range(len(x_orig))
                                    if y_orig[i] == 1])

# Negative Data Points
x_neg = np.array([x_orig[i] for i in range(len(x_orig))
                                    if y_orig[i] == 0])

# Plotting the Positive Data Points
```

```
plt.scatter(x_pos[:, 0], x_pos[:, 1], color = 'blue', label = 'Positive')

# Plotting the Negative Data Points
plt.scatter(x_neg[:, 0], x_neg[:, 1], color = 'red', label = 'Negative')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Plot of given data')
plt.legend()

plt.show()
```



In [5]:
```
oneHot = OneHotEncoder()

# Encoding x_orig
oneHot.fit(x_orig)
x = oneHot.transform(x_orig).toarray()

# Encoding y_orig
oneHot.fit(y_orig)
y = oneHot.transform(y_orig).toarray()

alpha, epochs = 0.0035, 500
m, n = x.shape
print('m =', m)
print('n =', n)
print('Learning Rate =', alpha)
print('Number of Epochs =', epochs)
```

```
m = 100
n = 51
Learning Rate = 0.0035
Number of Epochs = 500
```

In [6]:
```
X = tf.placeholder(tf.float32, [None, n])
```

Loading [MathJax]/extensions/Safe.js

```python
# Since this is a binary classification problem,
# Y can take only 2 values.
Y = tf.placeholder(tf.float32, [None, 2])

# Trainable Variable Weights
W = tf.Variable(tf.zeros([n, 2]))

# Trainable Variable Bias
b = tf.Variable(tf.zeros([2]))
```

In [7]:

```python
Y_hat = tf.nn.sigmoid(tf.add(tf.matmul(X, W), b))

# Sigmoid Cross Entropy Cost Function
cost = tf.nn.sigmoid_cross_entropy_with_logits(
                    logits = Y_hat, labels = Y)

# Gradient Descent Optimizer
optimizer = tf.train.GradientDescentOptimizer(
         learning_rate = alpha).minimize(cost)

# Global Variables Initializer
init = tf.global_variables_initializer()
```

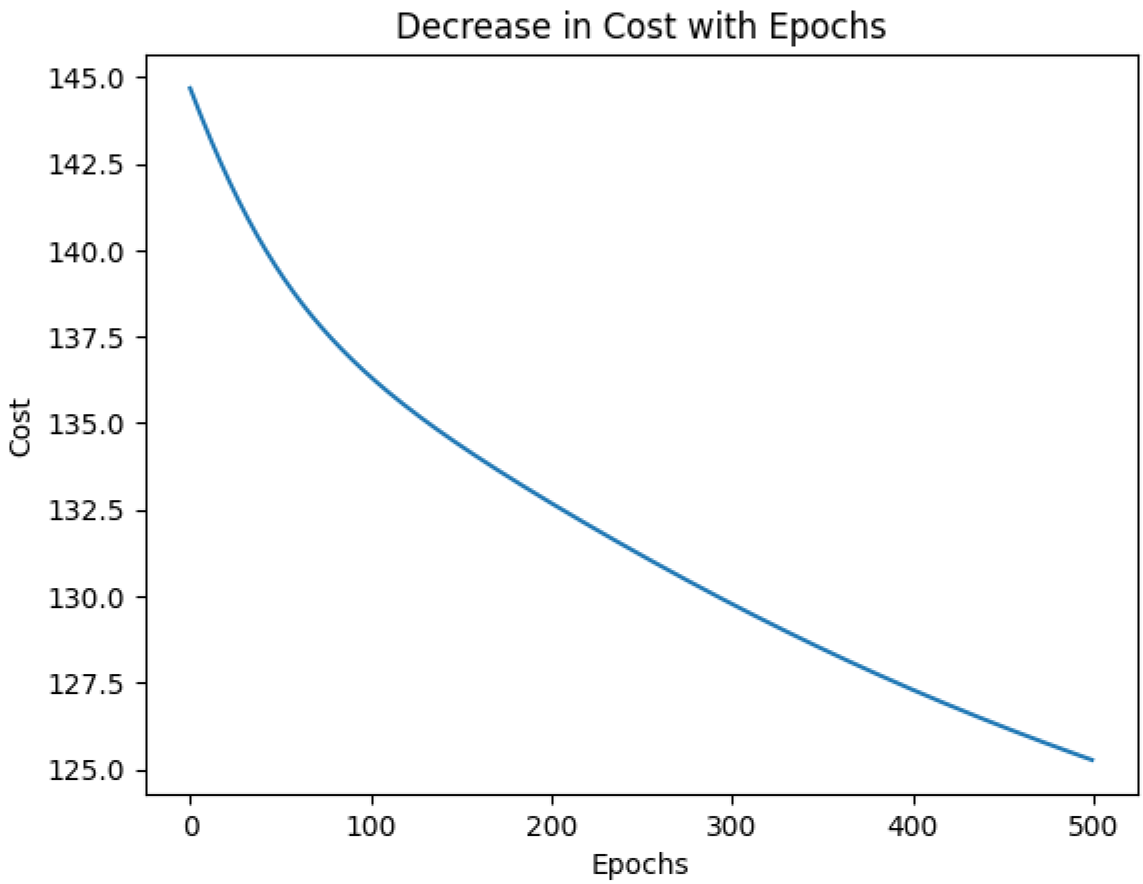
In [8]:

```python
with tf.Session() as sess:

    # Initializing the Variables
    sess.run(init)

    # Lists for storing the changing Cost and Accuracy in every Epoch
    cost_history, accuracy_history = [], []

    # Iterating through all the epochs
    for epoch in range(epochs):
        cost_per_epoch = 0

        # Running the Optimizer
        sess.run(optimizer, feed_dict = {X : x, Y : y})

        # Calculating cost on current Epoch
        c = sess.run(cost, feed_dict = {X : x, Y : y})

        # Calculating accuracy on current Epoch
        correct_prediction = tf.equal(tf.argmax(Y_hat, 1),
                                          tf.argmax(Y, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction,
                                                 tf.float32))

        # Storing Cost and Accuracy to the history
        cost_history.append(sum(sum(c)))
        accuracy_history.append(accuracy.eval({X : x, Y : y}) * 100)

        # Displaying result on current Epoch
        if epoch % 100 == 0 and epoch != 0:
            print("Epoch " + str(epoch) + " Cost: "
                            + str(cost_history[-1]))

    Weight = sess.run(W) # Optimized Weight
    Bias = sess.run(b)   # Optimized Bias

    # Final Accuracy
    correct_prediction = tf.equal(tf.argmax(Y_hat, 1),
                                      tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction,
```

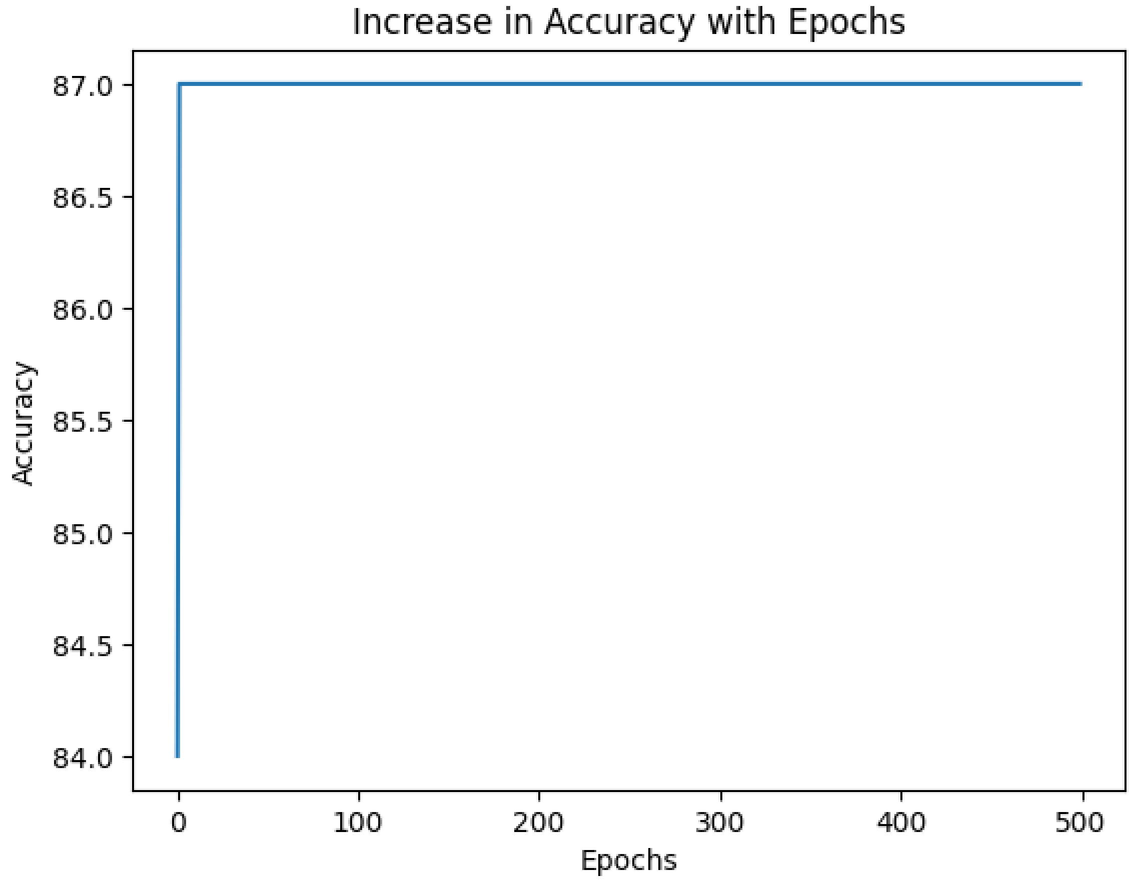
```
                                                 tf.float32))
    print("\nAccuracy:", accuracy_history[-1], "%")
```

```
Epoch 100 Cost: 136.33413696289062
Epoch 200 Cost: 132.68544006347656
Epoch 300 Cost: 129.771240234375
Epoch 400 Cost: 127.29396057128906

Accuracy: 87.00000047683716 %
```

In [9]:
```
plt.plot(list(range(epochs)), cost_history)
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Decrease in Cost with Epochs')

plt.show()
```



In [10]:
```
plt.plot(list(range(epochs)), accuracy_history)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Increase in Accuracy with Epochs')

plt.show()
```


Increase in Accuracy with Epochs
87.0
86.5
86.0
85.5
85.0
84.5
84.0
Accuracy
0
100
200
300
400
500
Epochs

Deep Neural Network

LAB 6: -Building a NN model with tensorflow.

Name: - Gaurav Sonawane

PRN: - 20200802154

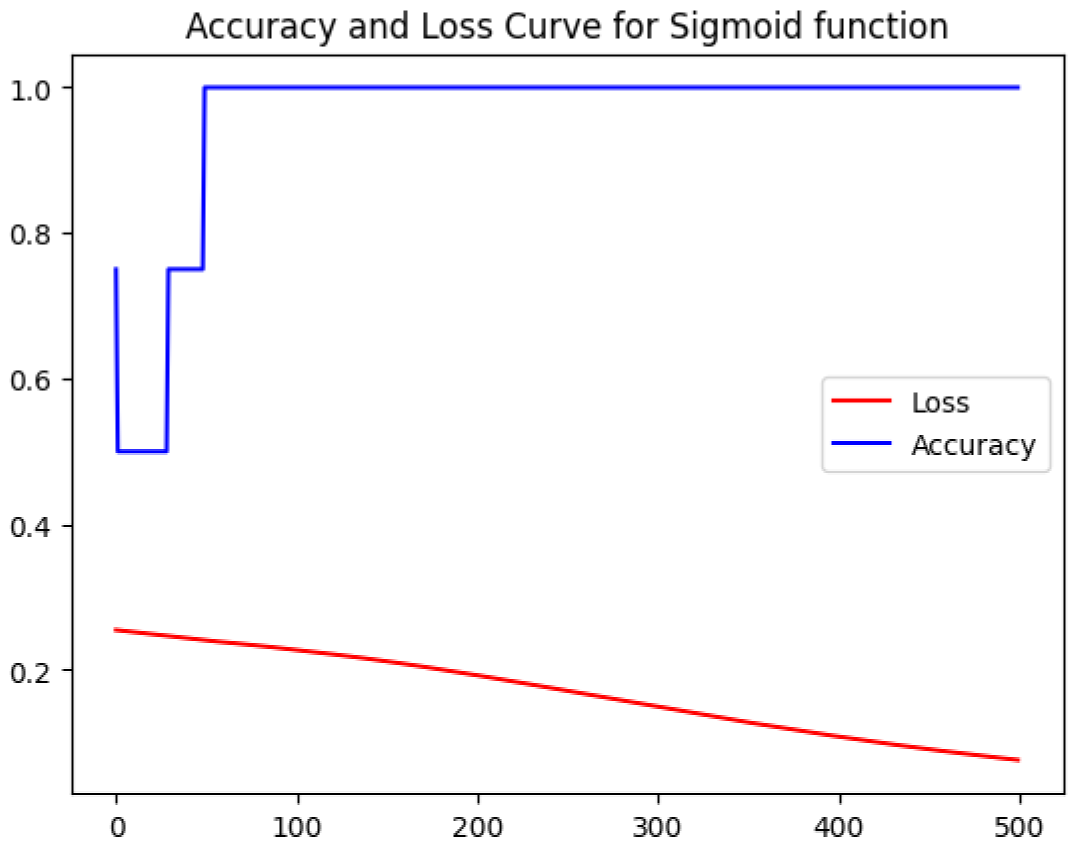
BTech CSE TY

DS1

In [2]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers.core import Dense
import matplotlib
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

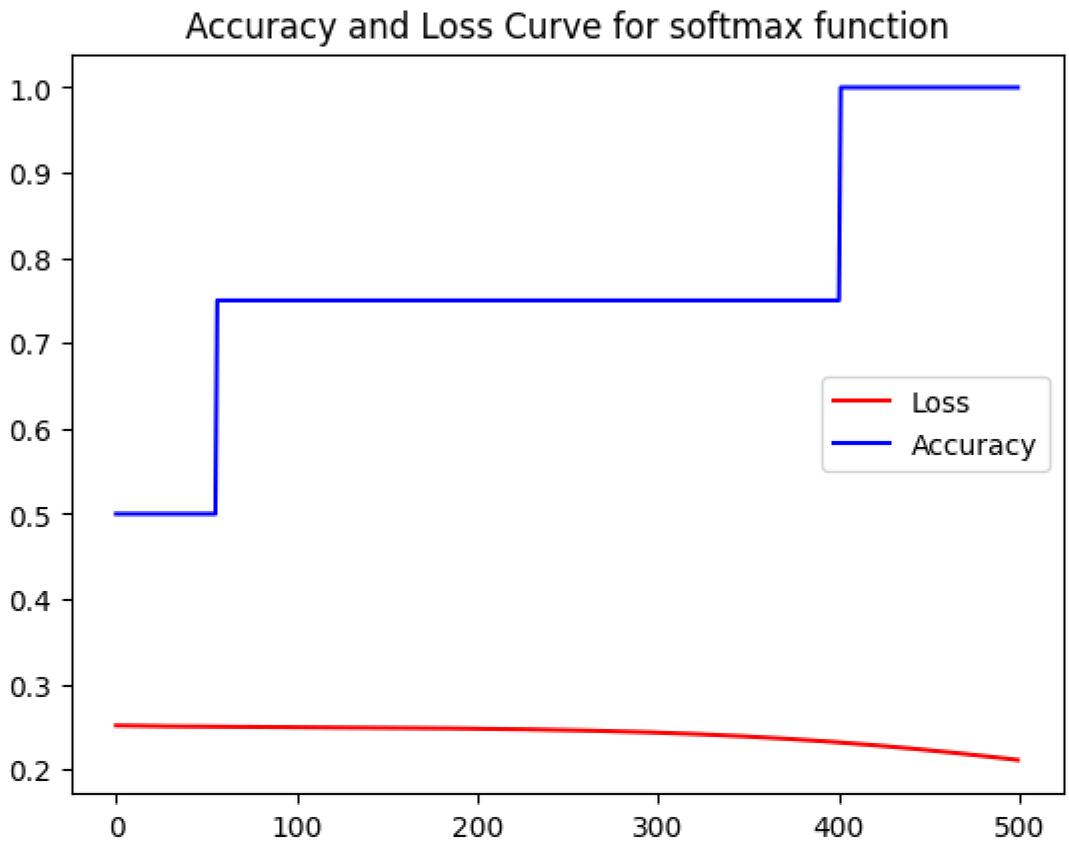
In [3]:
```
# the four different states of the XOR gate
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")

# the four expected results in the same order
target_data = np.array([[0],[1],[1],[0]], "float32")
```

In [4]:
```
model = Sequential()
model.add(Dense(16, input_dim=2, activation='relu'))
model.add(Dense(1, activation="sigmoid"))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])

history = model.fit(training_data, target_data, epochs=500, verbose=0)

loss_curve = history.history["loss"]
acc_curve = history.history["binary_accuracy"]

plt.plot(loss_curve, label="Loss", color="r")
plt.plot(acc_curve, label="Accuracy", color="b")
plt.legend(loc="center right")
plt.title(f"Accuracy and Loss Curve for Sigmoid function")
plt.show()
```



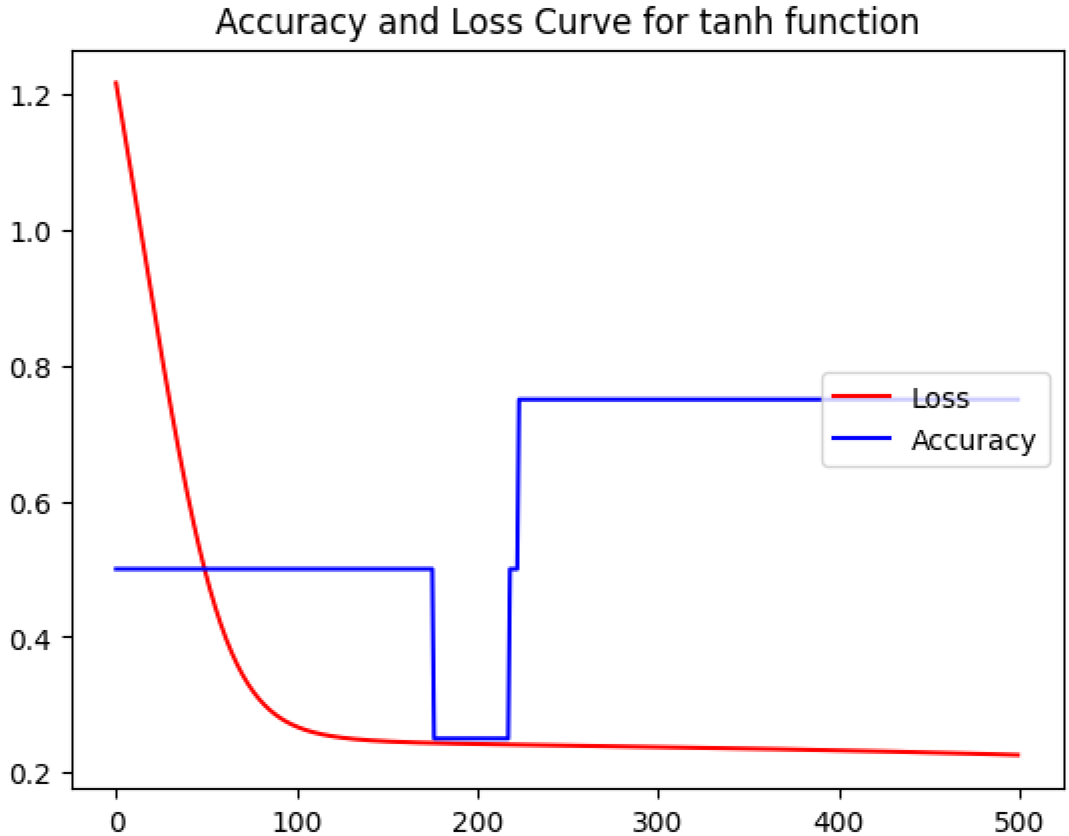
In [24]:

```
model = Sequential()
model.add(Dense(16, input_dim=2, activation='softmax'))
model.add(Dense(1, activation="sigmoid"))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])

history = model.fit(training_data, target_data, epochs=500, verbose=0)

loss_curve = history.history["loss"]
acc_curve = history.history["binary_accuracy"]

plt.plot(loss_curve, label="Loss", color="r")
plt.plot(acc_curve, label="Accuracy", color="b")
plt.legend(loc="center right")
plt.title(f"Accuracy and Loss Curve for softmax function")
plt.show()
```



In [19]:

```
model = Sequential()
model.add(Dense(16, input_dim=2, activation='tanh'))
model.add(Dense(1, activation="tanh"))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])

history = model.fit(training_data, target_data, epochs=500, verbose=0)

loss_curve = history.history["loss"]
acc_curve = history.history["binary_accuracy"]

plt.plot(loss_curve, label="Loss", color="r")
plt.plot(acc_curve, label="Accuracy", color="b")
plt.legend(loc="center right")
plt.title(f"Accuracy and Loss Curve for tanh function")
plt.show()
```


Accuracy and Loss Curve for tanh function
1.2
1.0
0.8
0.6
0.4
0.2
0
100
200
300
400
500
Loss
Accuracy

Deep Neural Network

LAB 7: -Implement forward propagation.

Name: - Gaurav Sonawane

PRN: - 20200802154

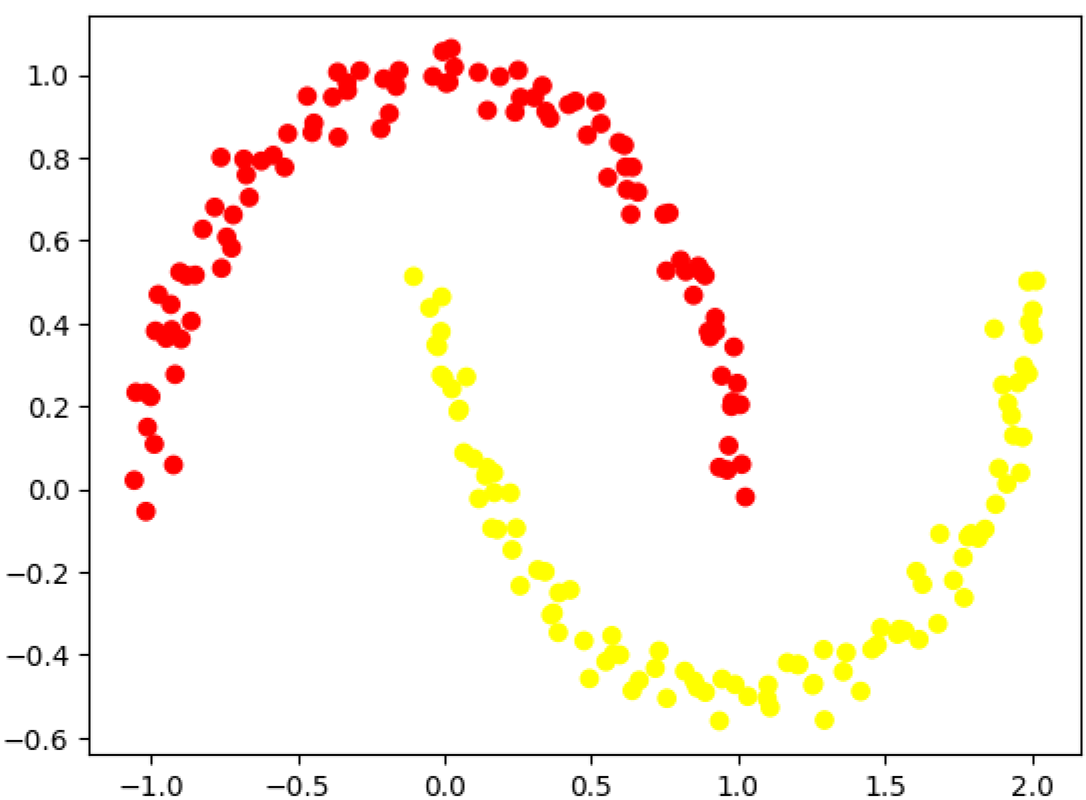
BTech CSE TY

DS1

In [1]:
```
#importing Required Libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors
from sklearn.datasets import make_moons
```

In [2]:
```
np.random.seed(0)
data, labels = make_moons(n_samples=200,noise = 0.04,random_state=0)
print(data.shape, labels.shape)
color_map = matplotlib.colors.LinearSegmentedColormap.from_list("", ["Red","yellow"])
plt.scatter(data[:,0], data[:,1], c=labels,cmap=color_map)
plt.show()
```

```
(200, 2) (200,)
```



In [3]:
```
from sklearn.model_selection import train_test_split
#Splitting the data into training and testing data
```

```
X_train, X_val, Y_train, Y_val = train_test_split(data, labels, stratify=labels, random_
print(X_train.shape, X_val.shape)
```

(150, 2) (50, 2)

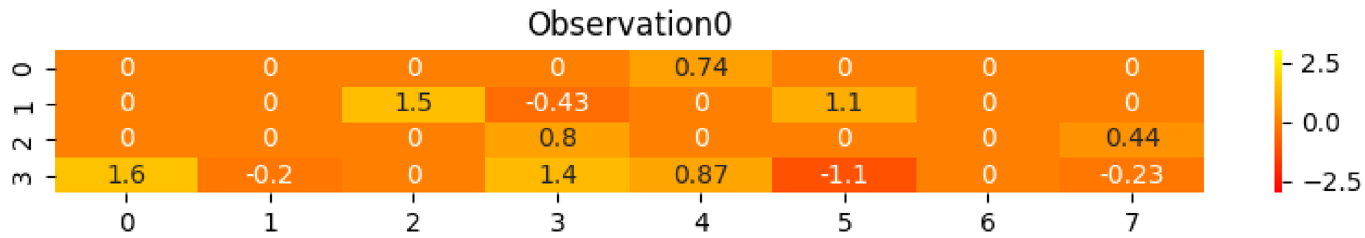
In [4]:
```
# Define a class for forward propagation where weights are randomly initialized.

class FeedForwardNetwork:

 def __init__(self):
     np.random.seed(0)
     self.w1 = np.random.randn()
     self.w2 = np.random.randn()
     self.w3 = np.random.randn()
     self.w4 = np.random.randn()
     self.w5 = np.random.randn()
     self.w6 = np.random.randn()
     self.b1 = 0
     self.b2 = 0
     self.b3 = 0

 def sigmoid(self, x):
     return 1.0/(1.0 + np.exp(-x))

 def forward_pass(self, x):
     self.x1, self.x2 = x
     self.a1 = self.w1*self.x1 + self.w2*self.x2 + self.b1
     self.h1 = self.sigmoid(self.a1)
     self.a2 = self.w3*self.x1 + self.w4*self.x2 + self.b2
     self.h2 = self.sigmoid(self.a2)
     self.a3 = self.w5*self.h1 + self.w6*self.h2 + self.b3
     self.h3 = self.sigmoid(self.a3)
     forward_matrix = np.array([[0,0,0,0,self.h3,0,0,0],
                  [0,0,(self.w5*self.h1),        (self.w6*self.h2),self.b3,self.a3,0
                  [0,0,0,self.h1,0,0,0,self.h2],
                  [(self.w1*self.x1), (self.w2*self.x2),self.b1, self.a1,(self.w3*se
     forward_matrices.append(forward_matrix)
     return self.h3
```

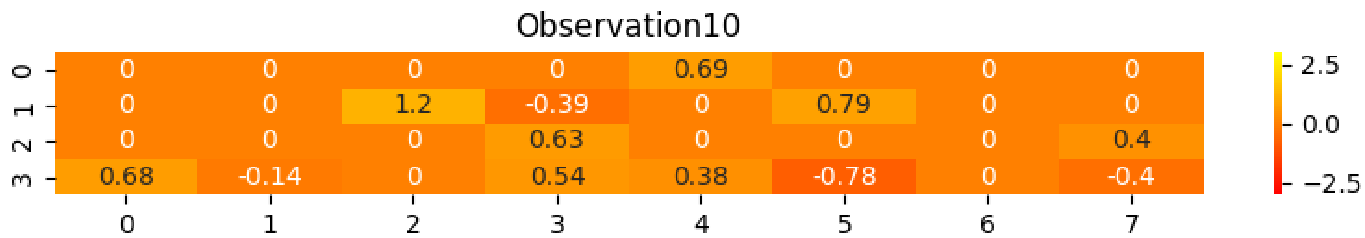
In [5]:
```
forward_matrices = []
ffn = FeedForwardNetwork()
for x in X_train:
   ffn.forward_pass(x)
```

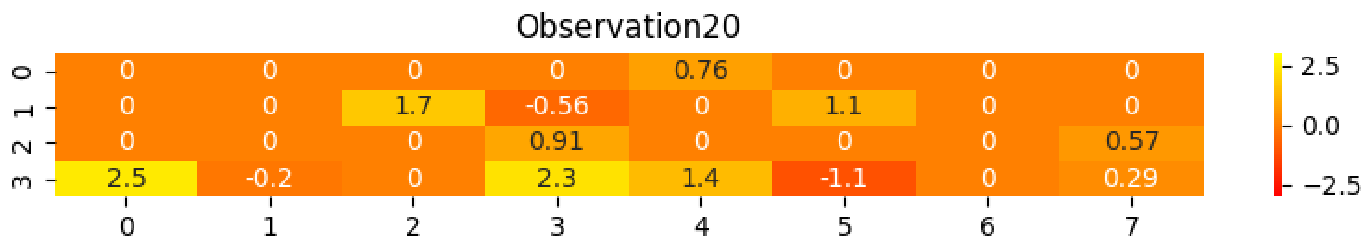
In [6]:
```
import seaborn as sns
import imageio
from IPython.display import HTML
def plot_heat_map(observation):
    fig = plt.figure(figsize=(10, 1))
    sns.heatmap(forward_matrices[observation], annot=True,     cmap= color_map, vmin=-3,
    plt.title("Observation"+str(observation))
    fig.canvas.draw()
    image = np.frombuffer(fig.canvas.tostring_rgb(), dtype='uint8')
    image = image.reshape(fig.canvas.get_width_height()[::-1] +             (3,))
    return image
imageio.mimsave([plot_heat_map(i) for i in range(0,len(forward_matrices),len(forward_mat
```

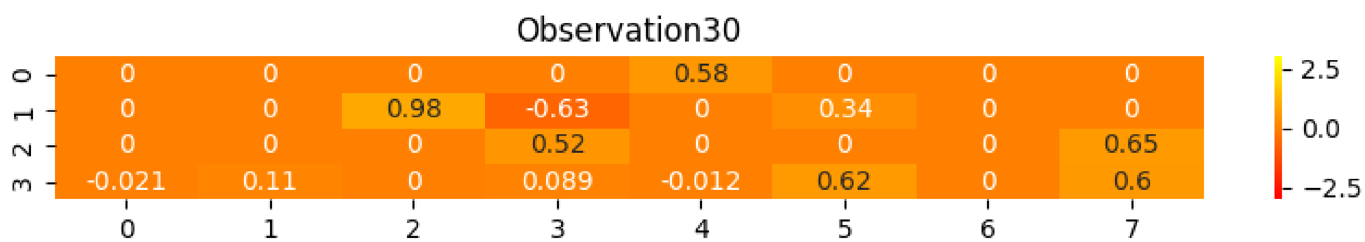
Loading [MathJax]/extensions/Safe.js

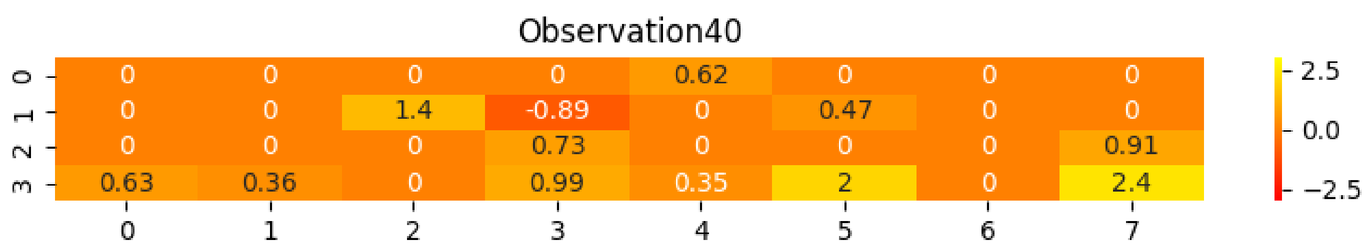
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-8210cda3ae8e> in <cell line: 12>()
     10     image = image.reshape(fig.canvas.get_width_height()[::-1] +
(3,))
     11     return image
---> 12 imageio.mimsave([plot_heat_map(i) for i in range(0,len(forward_matrices),len(for
ward_matrices)//15)], fps=1)

TypeError: mimwrite() missing 1 required positional argument: 'ims'
```

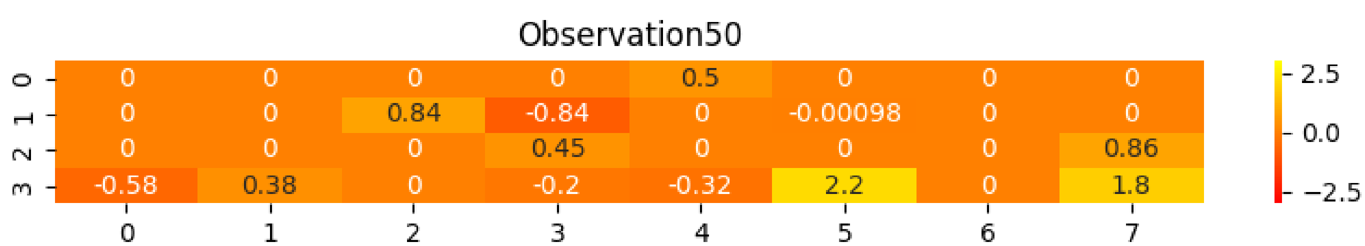


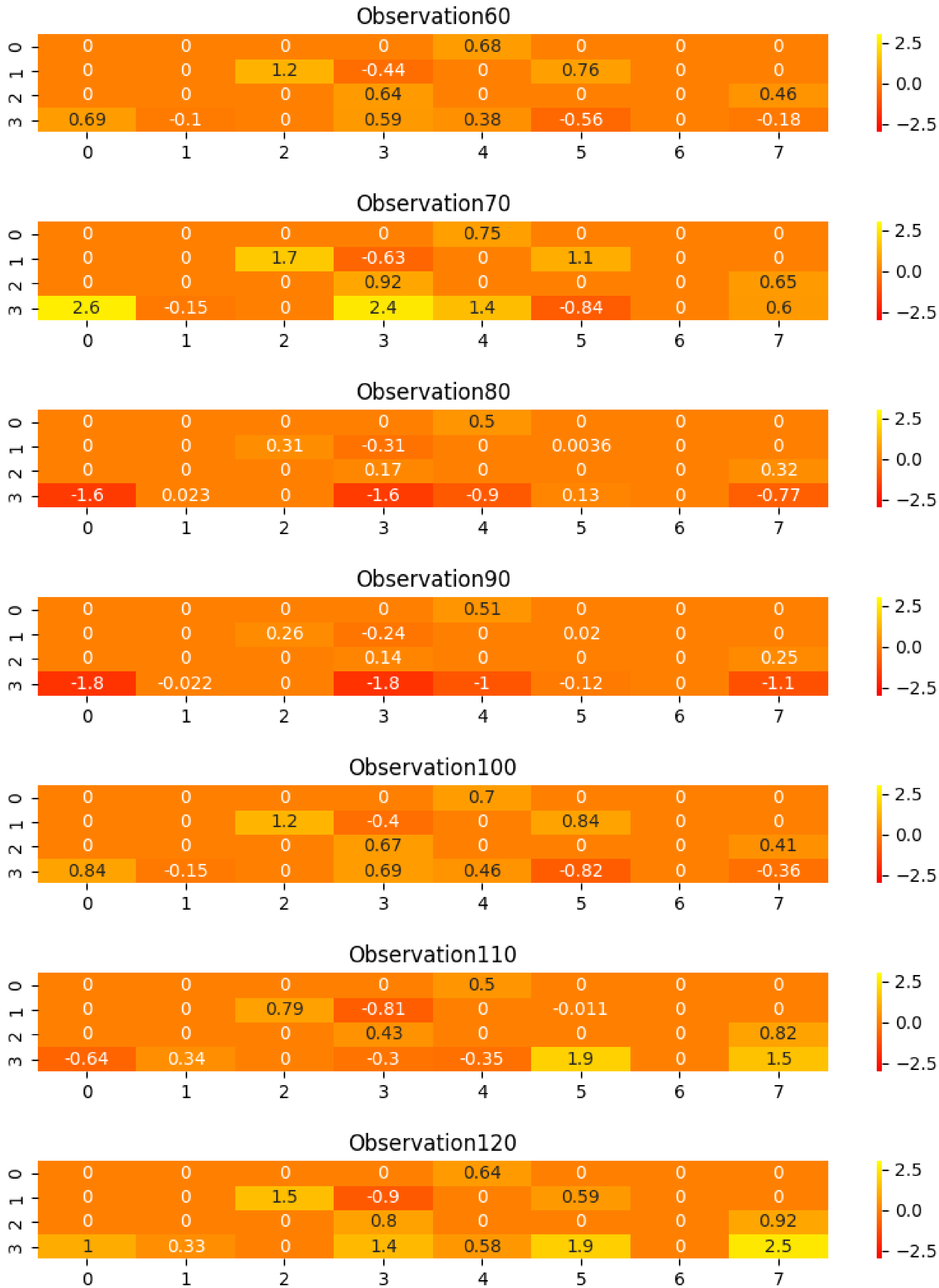


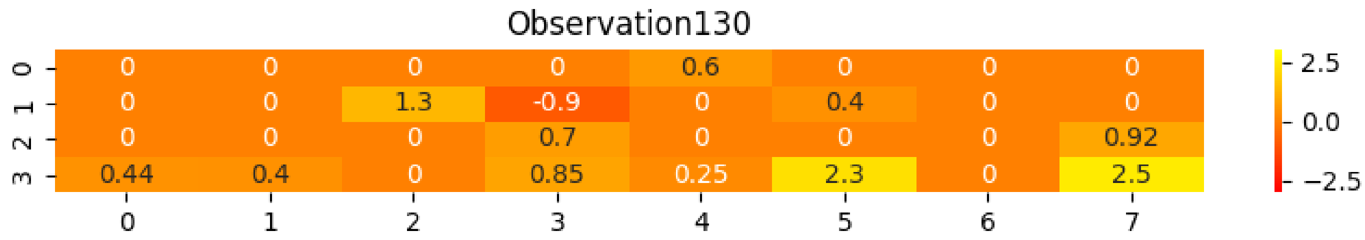


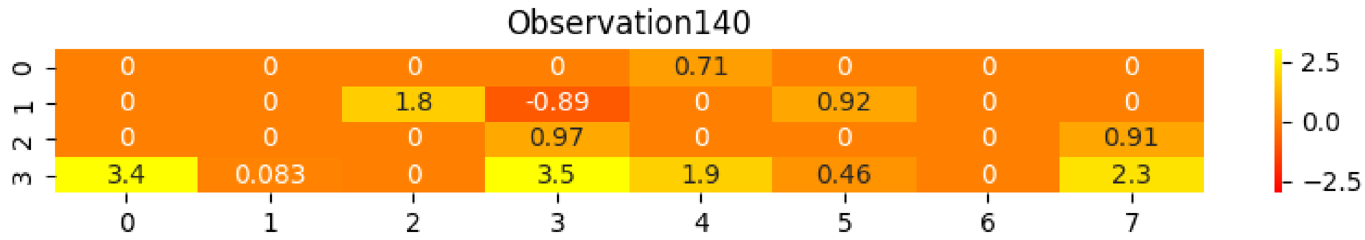







Observation60
Observation70
Observation80
Observation90
Observation100
Observation110
Observation120
2.5
0.0
−2.5





In [7]:

```
class FeedForwardNetwork_Vectorised:

 def __init__(self):
    np.random.seed(0)
    self.W1 = np.random.randn(2,2)
    self.W2 = np.random.randn(2,1)
    self.B1 = np.zeros((1,2))
    self.B2 = np.zeros((1,1))

 def sigmoid(self, X):
    return 1.0/(1.0 + np.exp(-X))


 def forward_pass(self,X):
    self.A1 = np.matmul(X,self.W1) + self.B1
    self.H1 = self.sigmoid(self.A1)
    self.A2 = np.matmul(self.H1, self.W2) + self.B2
    self.H2 = self.sigmoid(self.A2)
    return self.H2
ffn_v = FeedForwardNetwork_Vectorised()
ffn_v.forward_pass(X_train)
```

```
Out[7]: array([[0.74680041],
       [0.70022634],
       [0.48421485],
       [0.6470818 ],
       [0.76044078],
       [0.72492131],
       [0.58259395],
       [0.68175478],
       [0.58553591],
       [0.72745957],
       [0.67948245],
       [0.45878414],
       [0.75358062],
       [0.77924177],
       [0.55488996],
       [0.5691662 ],
       [0.64192607],
       [0.78055957],
       [0.72746492],
       [0.46516146],
       [0.78354368],
       [0.46973463],
       [0.71993656],
       [0.77610631],
       [0.74481201],
       [0.65331827],
       [0.69800807],
       [0.67358968],
       [0.76396575],
       [0.61886848],
       [0.60237268],
       [0.70134711],
       [0.44542862],
       [0.76511399],
       [0.75463325],
       [0.46317533],
       [0.7022096 ],
       [0.60417209],
       [0.73012384],
       [0.76953777],
       [0.65792742],
       [0.64128292],
       [0.77931947],
       [0.7721725 ],
       [0.64519697],
       [0.67004602],
       [0.76924104],
       [0.69657271],
       [0.78083278],
       [0.63300097],
       [0.55870697],
       [0.47550834],
       [0.45880048],
       [0.61429152],
       [0.6387825 ],
       [0.68349432],
       [0.57506716],
       [0.67958347],
       [0.7089558 ],
       [0.74223065],
       [0.6782777 ],
       [0.63822004],
       [0.67504992],
       [0.67620417],
```

```
[0.61024132],
[0.7381461 ],
[0.60154991],
[0.45876889],
[0.72657098],
[0.65726326],
[0.77895915],
[0.75577962],
[0.70838066],
[0.51455774],
[0.73987557],
[0.5924023 ],
[0.62986719],
[0.67256843],
[0.59983504],
[0.76585595],
[0.47265451],
[0.6177898 ],
[0.61713256],
[0.72710605],
[0.7012157 ],
[0.76485302],
[0.652397  ],
[0.73550774],
[0.68001266],
[0.73999719],
[0.47296892],
[0.77476879],
[0.76740531],
[0.48545259],
[0.7765677 ],
[0.6218701 ],
[0.50950537],
[0.75321854],
[0.76694632],
[0.45287446],
[0.69292258],
[0.61361061],
[0.67526706],
[0.53076279],
[0.73753669],
[0.49507063],
[0.45382425],
[0.77475385],
[0.70433724],
[0.53180961],
[0.54722999],
[0.71054658],
[0.59480762],
[0.44852797],
[0.77863888],
[0.45813372],
[0.69137941],
[0.73261598],
[0.50308796],
[0.66455837],
[0.67820134],
[0.77242308],
[0.76564457],
[0.58181088],
[0.76614285],
[0.45683187],
[0.75140662],
[0.55939833],
```

```
[0.47459955],
[0.7148063 ],
[0.64883685],
[0.74365404],
[0.73230448],
[0.65679917],
[0.69360367],
[0.44836489],
[0.63249233],
[0.72038926],
[0.73862607],
[0.65122808],
[0.74283336],
[0.60916411],
[0.45648454],
[0.65669436],
[0.72788056],
[0.64815035],
[0.61783118],
[0.44941327],
[0.48475405],
[0.60191814]])
```

Deep Neural Network

LAB 8: -Implement Backward propagation.

Name: - Gaurav Sonawane

PRN: - 20200802154

BTech CSE TY

DS1

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
data = load_iris()

# Get features and target
X=data.data
y=data.target
```

In [3]:
```
y = pd.get_dummies(y).values

y[:3]
```

Out[3]:
```
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0]], dtype=uint8)
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)
```

In [5]:
```
learning_rate = 0.1
iterations = 5000
N = y_train.size

# number of input features
input_size = 4

# number of hidden layers neurons
hidden_size = 2

# number of neurons at the output layer
output_size = 3

results = pd.DataFrame(columns=["mse", "accuracy"])
```

In [6]:
```
np.random.seed(10)

g weight for the hidden layer
```

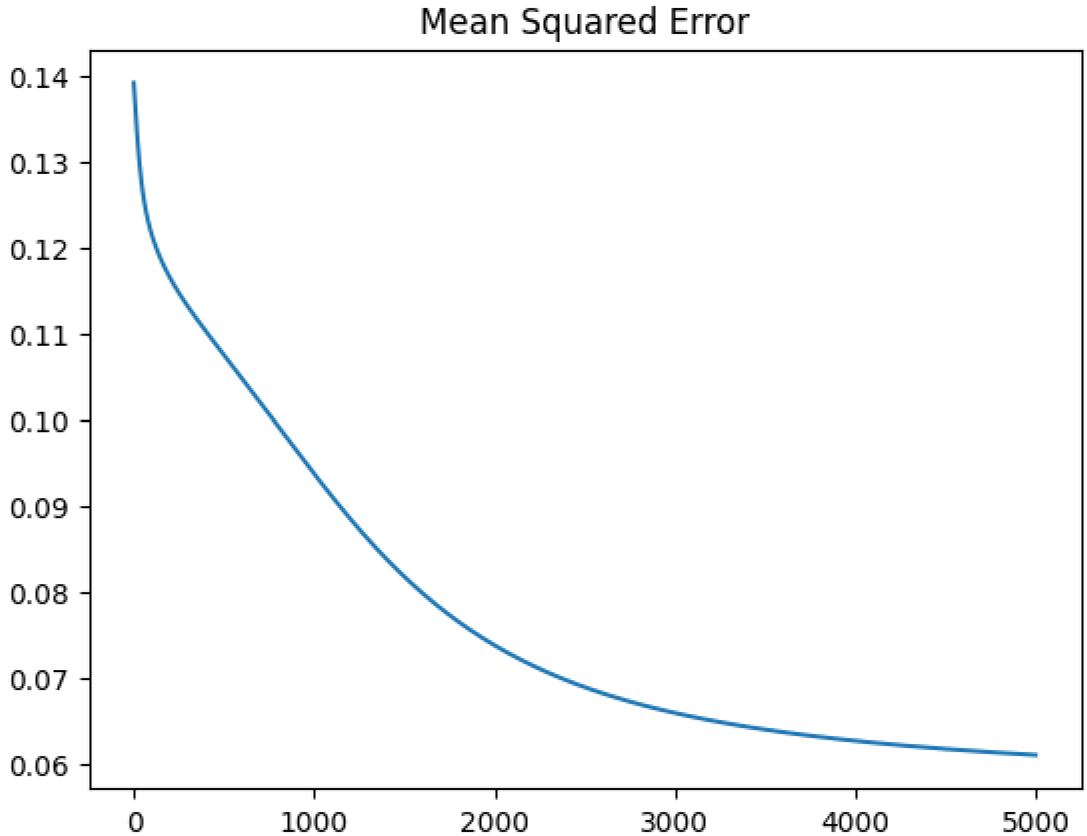
```python
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))

# initializing weight for the output layer
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))
```

In [7]:
```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true)**2).sum() / (2*y_pred.size)

def accuracy(y_pred, y_true):
    acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()
```

In [8]:
```python
for itr in range(iterations):

    # feedforward propagation
    # on hidden layer
    Z1 = np.dot(X_train, W1)
    A1 = sigmoid(Z1)

    # on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)


    # Calculating error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
    results=results.append({"mse":mse, "accuracy":acc},ignore_index=True )

    # backpropagation
    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)

    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)


    # weight updates
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(X_train.T, dW2) / N

    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update
```

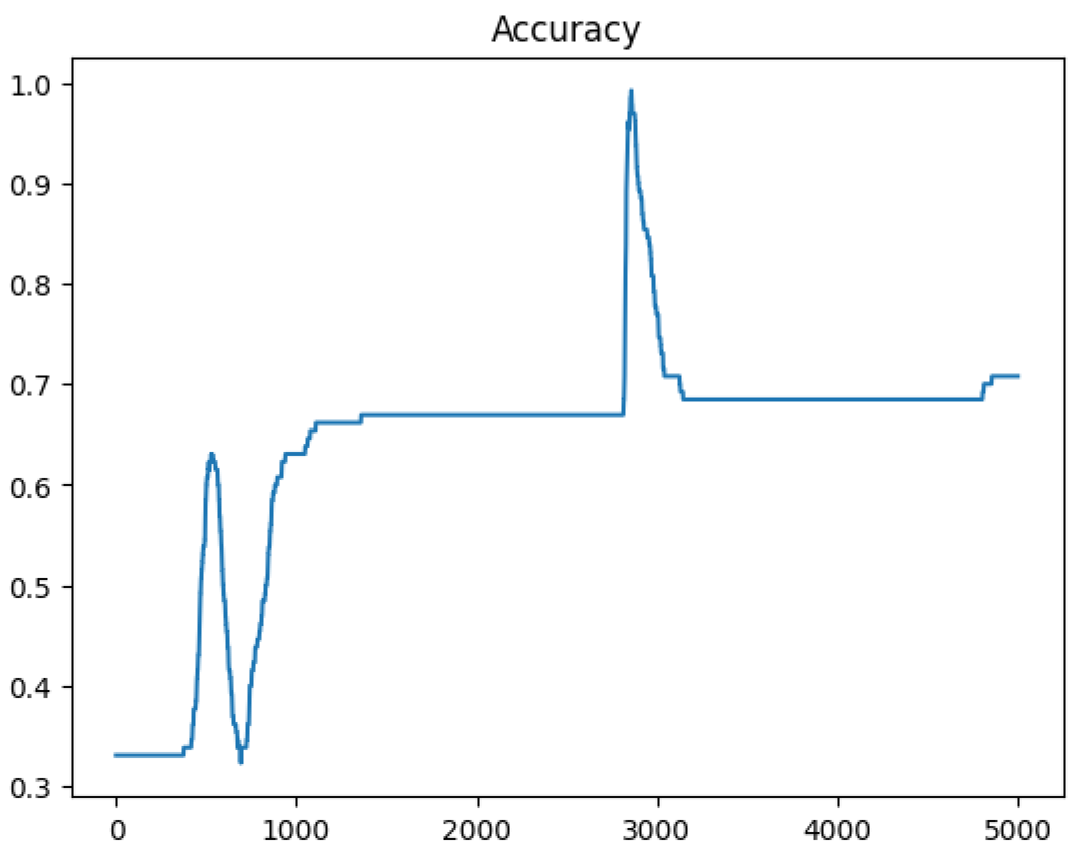
In [9]:
```python
results.mse.plot(title="Mean Squared Error")
```

Out[9]:
```
<Axes: title={'center': 'Mean Squared Error'}>
```



In [10]:
```
results.accuracy.plot(title="Accuracy")
```

Out[10]:
```
<Axes: title={'center': 'Accuracy'}>
```



In [11]:
```
Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)
```

```
acc = accuracy(A2, y_test)
print("Accuracy: {}".format(acc))
```

```
Accuracy: 0.8
```